Algorithms and Data Structures 2
CS 1501

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Homework 8 and Lab 7: Tuesday 3/21 @ 11:59 pm
  - Homework 9: this Friday @ 11:59 pm
  - Assignment 3: Friday 3/31 @ 11:59 pm
    - Support video and slides will be on Canvas
    - Debugging tips

Previous lecture

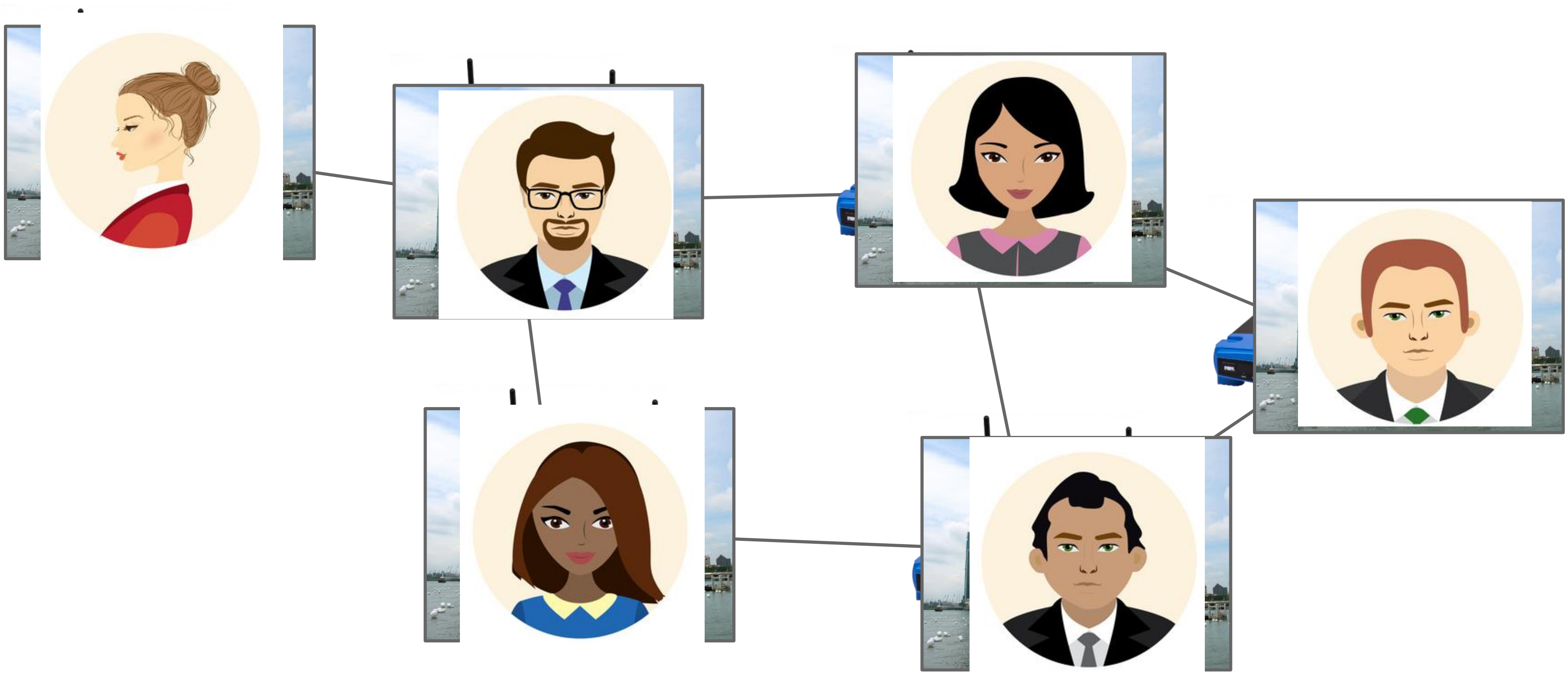
- Burrows-Wheeler Compression Algorithm
- ADT Graph
  - definitions
  - representation using array of linked lists

This Lecture

- ADT Graph
  - definitions
  - representations
    - two-arrays
    - adjacency matrix
    - adjacency lists
  - traversals
    - BFS
      - shortest paths based on number of edges
      - connected components
    - DFS
      - finding articulation points of a graph

Why?

- Can be used to model many different scenarios



Some definitions

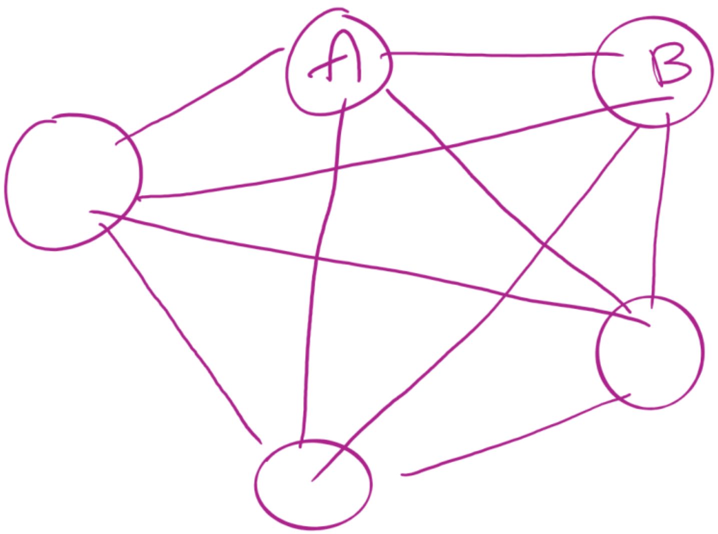
- Undirected graph
  - Edges are unordered pairs: (A, B) == (B, A)
- Directed graph
  - Edges are ordered pairs: (A, B) != (B, A)
- Adjacent vertices, or neighbors
  - Vertices connected by an edge

Graph sizes

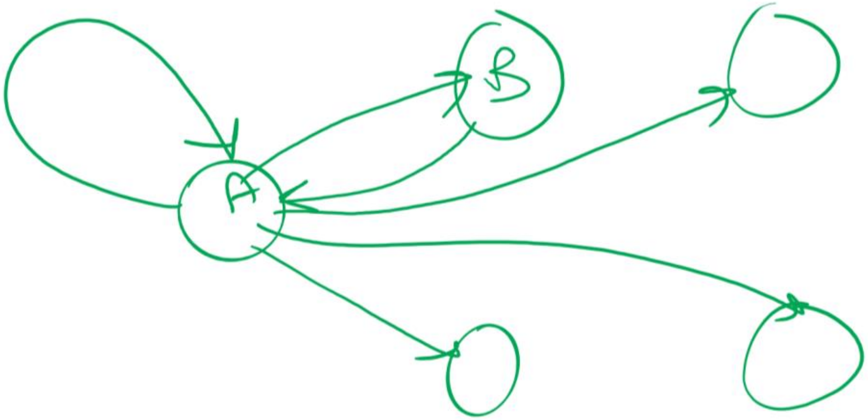
- Let v = |V|, and e = |E|
- Given v, what are the minimum/maximum sizes of e?
  - Minimum value of e?
    - Definition doesn't necessitate that there are any edges…
    - So, 0
  - Maximum of e?
    - Depends…
      - Are self edges allowed?
      - Directed graph or undirected graph?
    - In this class, we'll assume directed graphs have self edges while undirected graphs do not

Maximum value of e (MAX)

- Undirected graph
  - no self edges
  - v*(v-1)?
  - But, A->B is the same edge as B-> A
  - Are we counting each twice?
  - v*(v-1)/2

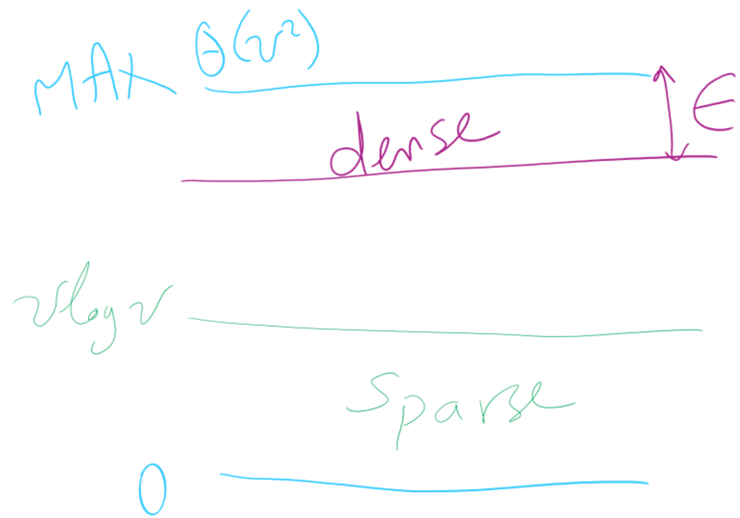


- Directed graph
  - self edges allowed
  - v*v?
  - A -> B is a different edge than B -> A
  - v^2

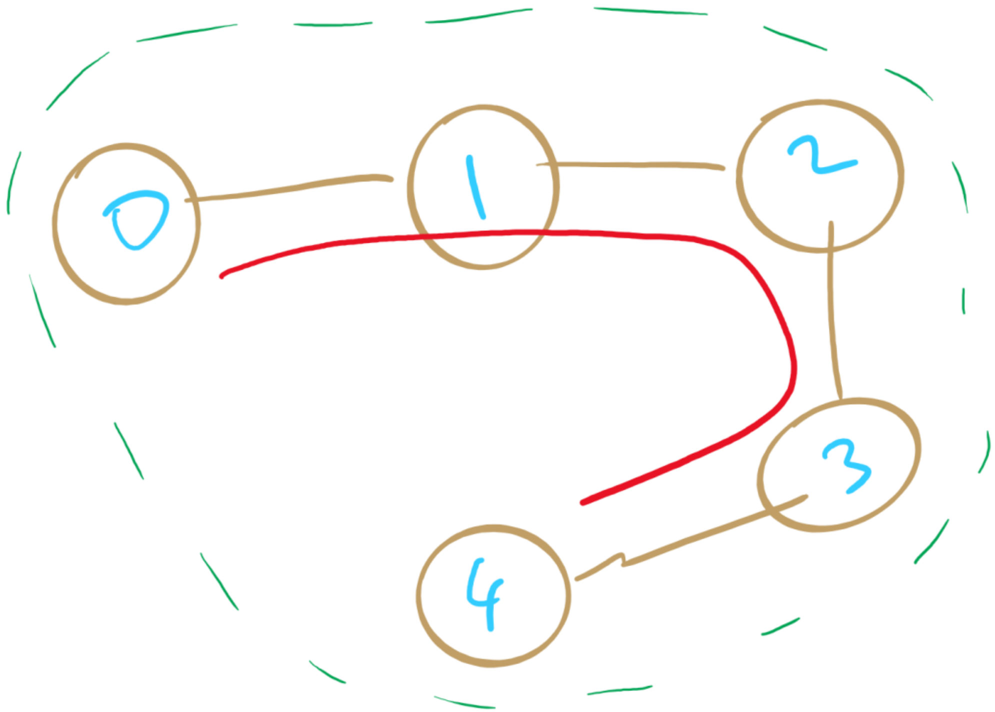


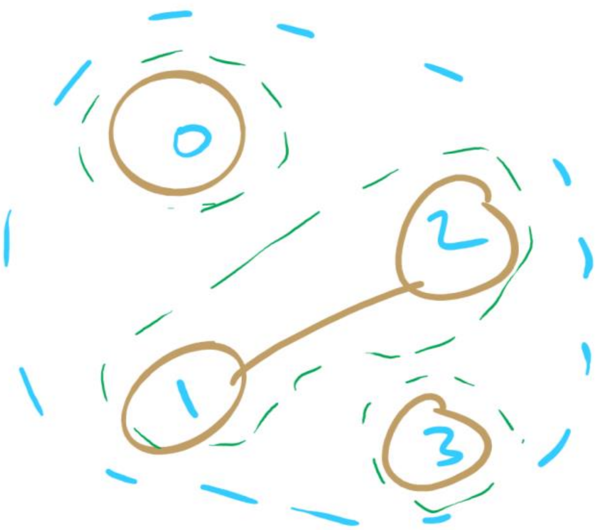
More definitions

- A graph is considered *sparse* if:
  - e <= v lg v
- A graph is considered *dense* as it approaches the maximum number of edges
  - I.e., e == MAX - ε
- A *complete* graph has the maximum number of edges
- Have we seen “sparse” and dense before?



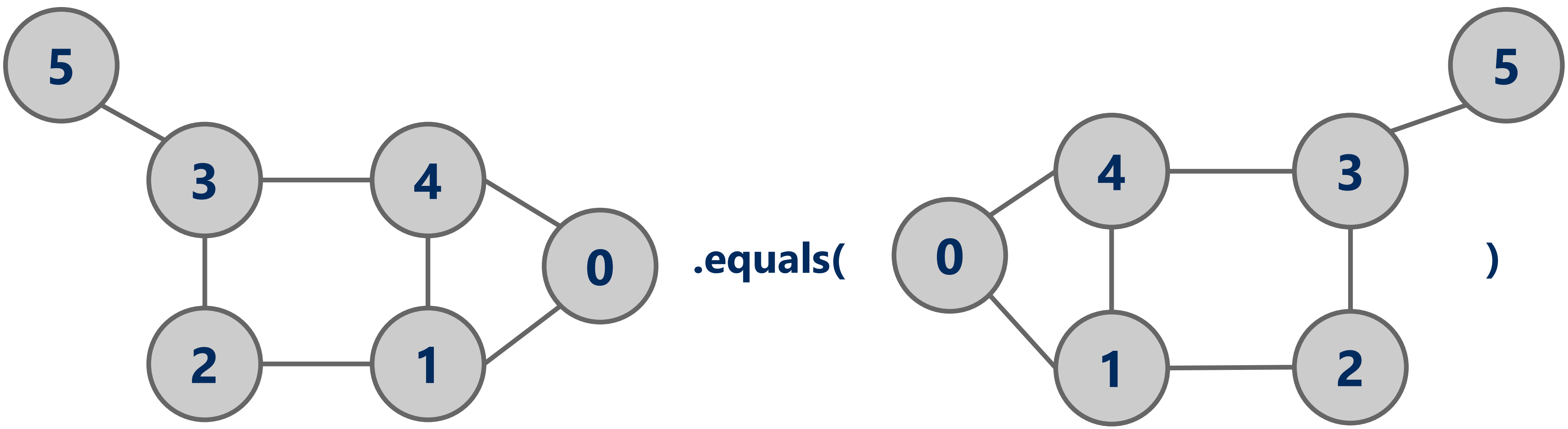
Sparse graphs





Question:

- Is



Representing graphs

- Trivially, graphs can be represented as:
  - List of vertices
  - List of edges
- Performance?
  - Assume we're going to be analyzing static graphs
    - I.e., no insert and remove
  - So what operations should we consider?

Graph operations

- Static graphs
  - check if two vertices are neighbors
  - find the list of neighbors of a given vertex
    - for directed graphs, in-neighbors and out-neighbors
- Dynamic graphs
  - add/remove edges
  - Not our focus in this class

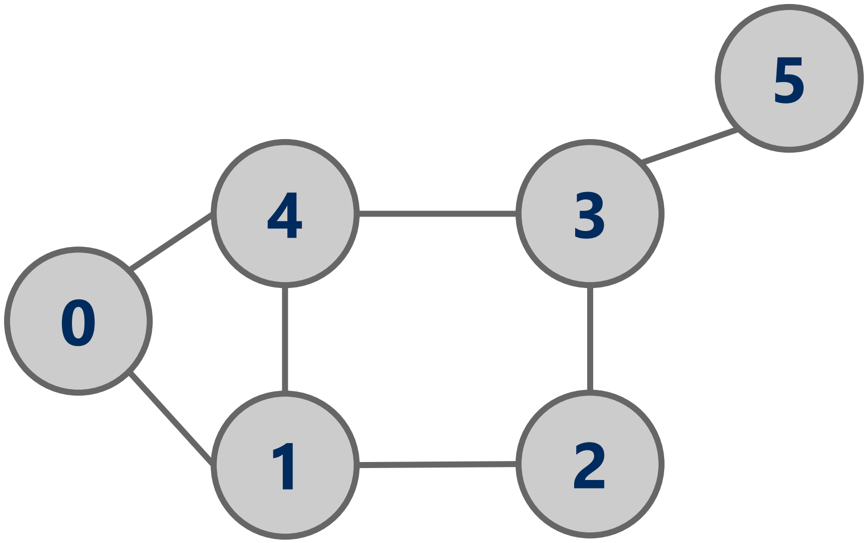
Representing graphs

- Trivially, graphs can be represented as:
    - List of vertices
    - List of edges
- Performance?
    - Check if two vertices are neighbors
        - O(e)
    - Find the list of neighbors of a given vertex
        - O(e)
- Space?
    - $\Theta(v + e)$ memory

Using an adjacency matrix

- Rows/columns are vertex labels
  - M[i][j] = 1 if (i, j) ∈ E
  - M[i][j] = 0 if (i, j) ∉ E

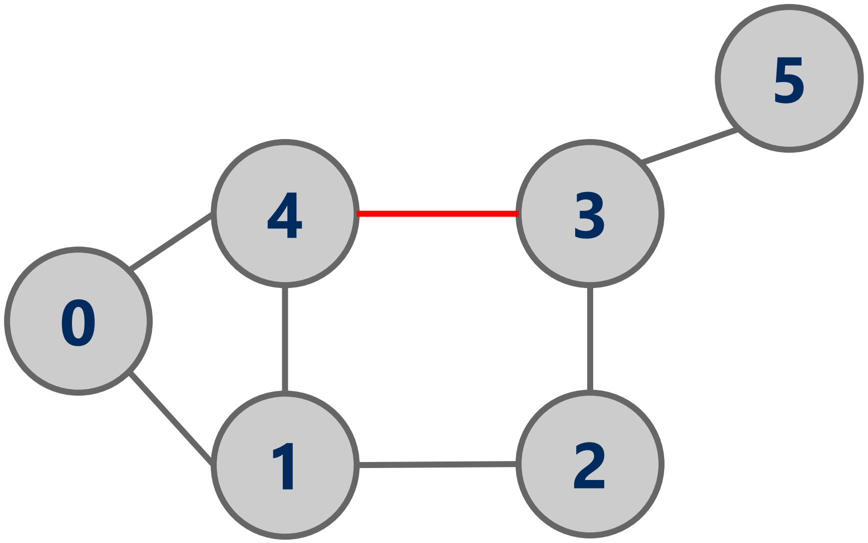
	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	0	1	0
2	0	1	0	1	0	0
3	0	0	1	0	1	1
4	1	1	0	1	0	0
5	0	0	0	1	0	0



Using an adjacency matrix

- Rows/columns are vertex labels
  - $M[i][j] = 1$ if $(i, j) \in E$
  - $M[i][j] = 0$ if $(i, j) \notin E$

	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	0	1	0
2	0	1	0	1	0	0
3	0	0	1	0	1	1
4	1	1	0	1	0	0
5	0	0	0	1	0	0



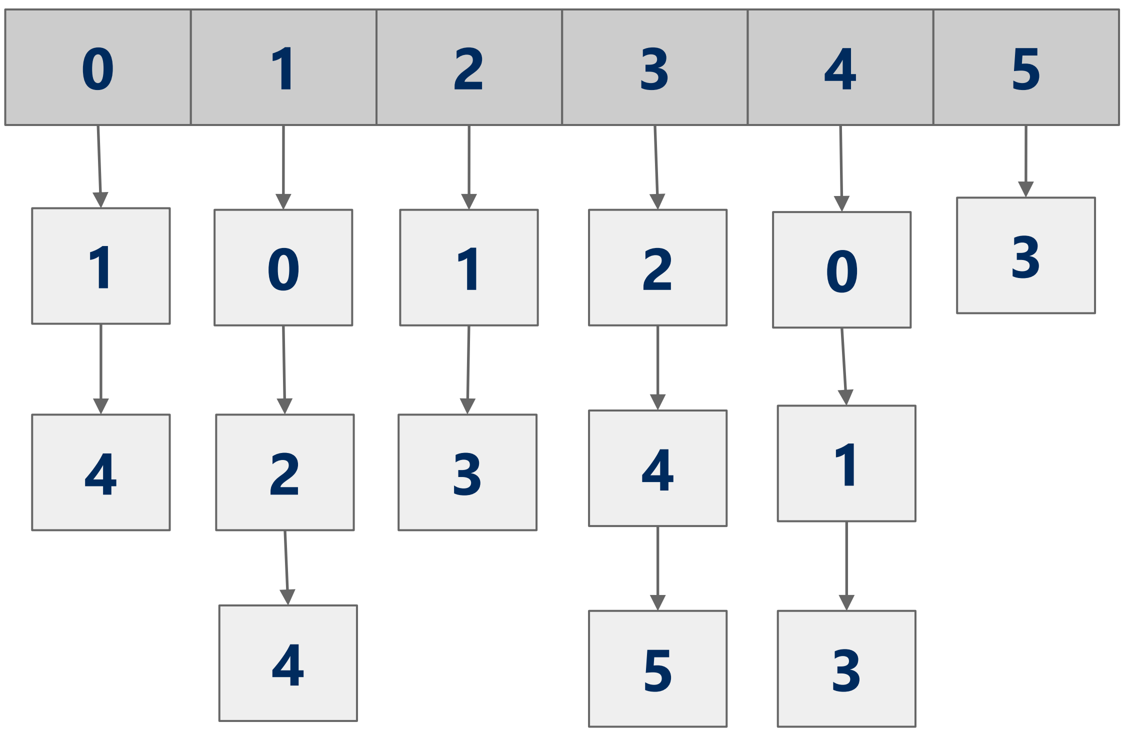
Adjacency matrix analysis

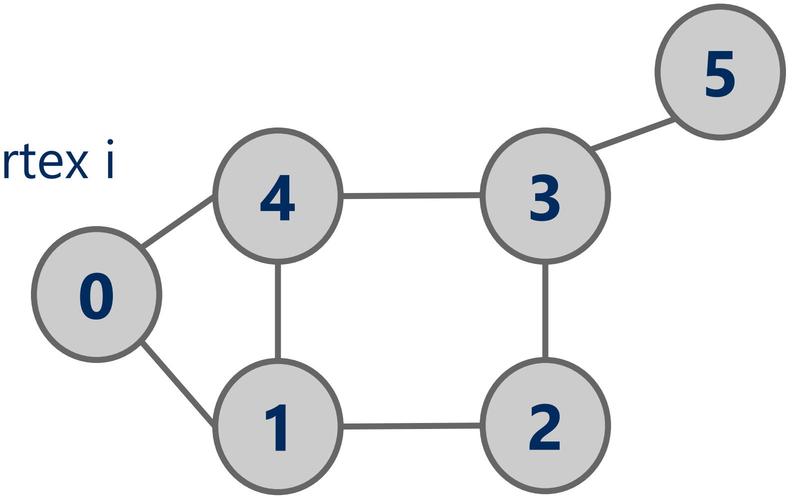
- Runtime?
  - Check if two vertices are neighbors
    - $\Theta(1)$
  - Find the list of neighbors of a vertex
    - $O(v)$
  - $O(v^2)$ time to initialize
- Space?
  - $O(v^2)$

	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	0	1	0
2	0	1	0	1	0	0
3	0	0	1	0	1	1
4	1	1	0	1	0	0
5	0	0	0	1	0	0

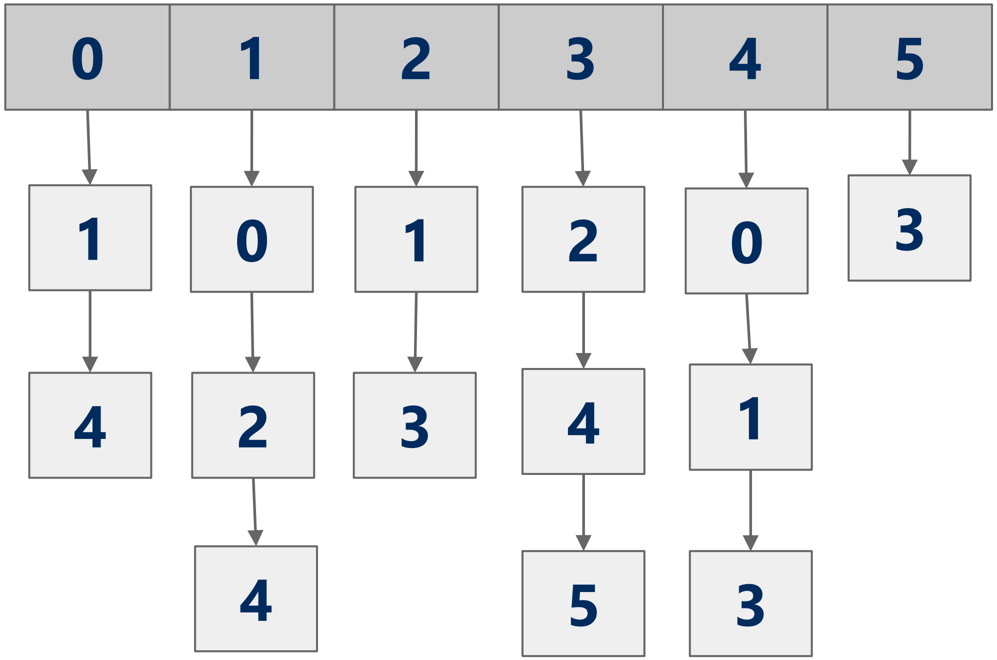
Adjacency lists

- Array of neighbor lists
  - A[i] contains a list of the neighbors of vertex i





Adjacency list analysis



- Runtime?
    - Check if two vertices are neighbors
    - Find the list of neighbors of a vertex
        - Θ(d)
        - d is the degree of a vertex (# of neighbors)
        - O(v)
- Space?
    - Θ(v + e) memory
    - overhead of node use
    - Could be much less than v^2

Comparison

- **Where would we want to use adjacency lists vs adjacency matrices?**
- Dense graphs?
- Sparse graphs?
- **What about the list of vertices/list of edges approach?**

Even more definitions

- Path
  - A sequence of adjacent vertices
- Simple Path
  - A path in which no vertices are repeated
- Simple Cycle
  - A simple path with the same first and last vertex
- Connected Graph
  - A graph in which a path exists between all vertex pairs
- Connected Component
  - Connected subgraph of a graph
- Acyclic Graph
  - A graph with no cycles
- Tree
  - ?
  - A connected, acyclic graph
    - Has exactly v-1 edges

Complete Graph vs. Connected Graph

- Difference between Connected graph and Complete graph?
  - Connected means there is a **path** from A to B for each pair of vertices A and B
  - Complete means there is an **edge** between A and B for each pair of vertices A and B

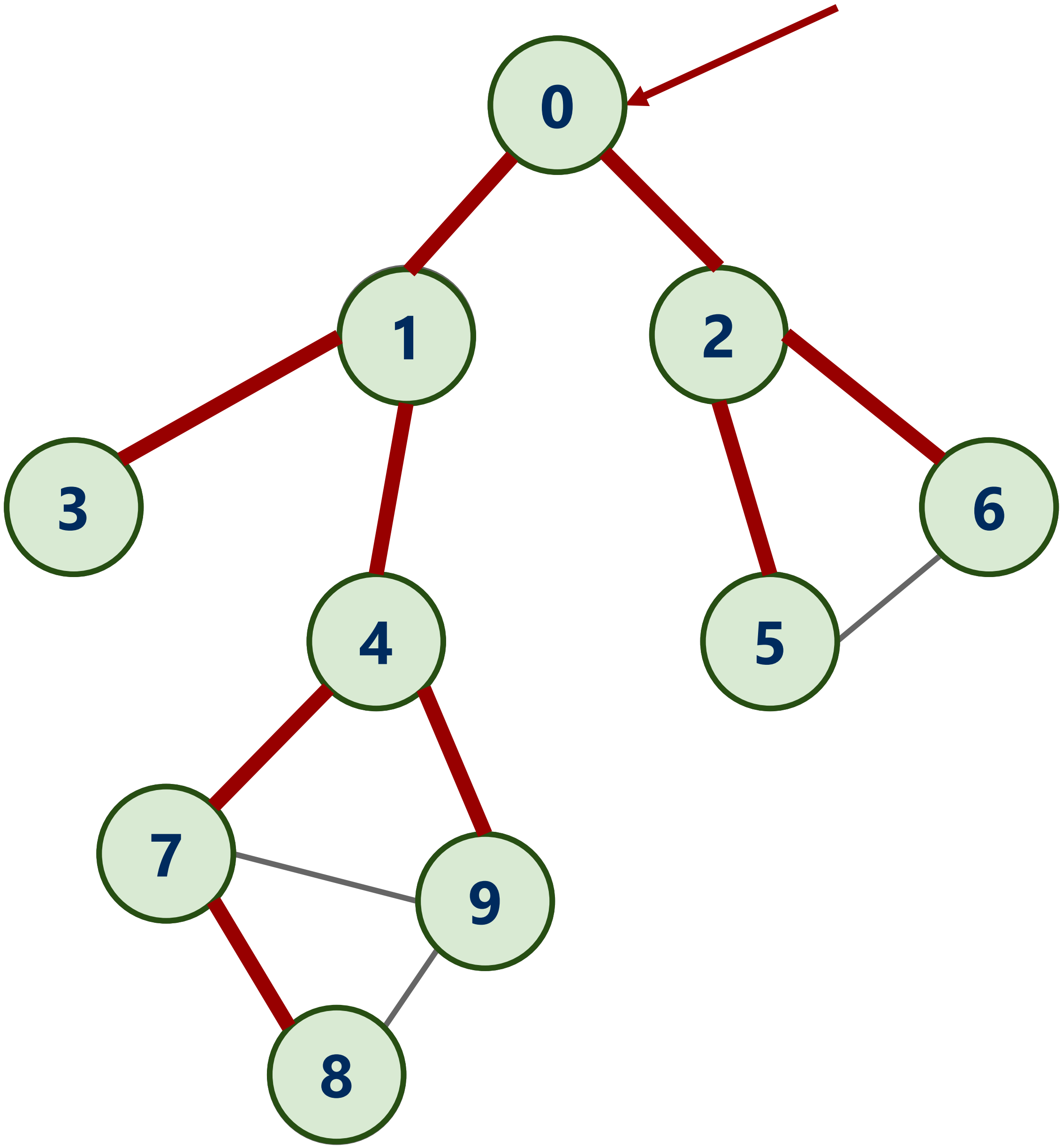
Graph traversal

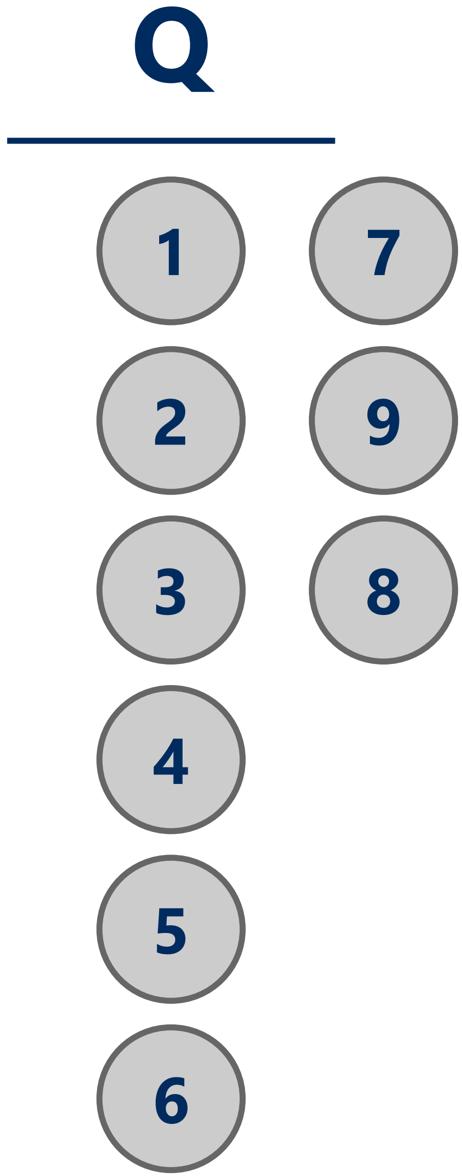
- What is the best order to traverse a graph?
- Two primary approaches:
  - Breadth-first search (BFS)
    - Search all directions evenly
      - i.e., from i, visit all of i's neighbors, then all of their neighbors, etc.
    - Would help us compute the distance between two vertices
      - Remember our Problem of the Day?
  - Depth-first search (DFS)
    - "Dive" as deep as possible into the graph first
    - Branch when necessary

BFS

- Can be easily implemented using a queue
  - For each vertex visited, add all of its neighbors to the Q (if not previously added)
    - Vertices that have been seen (i.e., added to the Q) but not yet visited are said to be the *fringe*
  - Pop head of the queue to be the next visited vertex
- See example

BFS example





BFS Pseudo-code

```
Q = new Queue

BFS(vertex v){
    add v to Q
    while(Q is not empty){
        w = remove head of Q
        visited[w] = true //mark w as visited
        for each unseen neighbor x
            seen[x] = true //mark x as seen
            parent[x] = w
            add x to Q
    }
}
```